AN ADAPTIVE UTILIZATION OF CONVOLUTIONAL MATRIX METHODS ON SLICED HIPPOCAMPAL NEURON CELL SEGMENTATION WITH AN APPLICATION INTERFACE

Neeraj Rattehalli[1] and Ishan Jain[2]

[1]Computer Science, Menlo-Atherton High School,
Atherton, California, , USA
[2]Computer Science, Mission San Jose High School,
Fremont, California, USA

ABSTRACT

Current methods of image analysis and segmentation on hippocampal neuron bodies contain excess and unwanted information like unnecessary noise. In order to clearly analyze each neural stain like DAPI, Cy5, TRITC, FITC and start the segmentation process, it is pertinent to preemptively denoise the data and create masked regions that accurately capture the ROI in these hippocampal regions. Unlike traditional edge detection algorithms like the Canny methods available in OpenCv libraries, we employed a more targeted approach based on pixel color intensities to segment out hippocampal neurons from the background. Using the R, G, and B value thresholds, our algorithm checks if a cell is a boundary point by doing neighboring pixel level comparisons. Combined with a seamless GUI interface for cropping the highlighted ROI, the algorithms efficiently work at creating general outlines of neuron bodies. With user modularity from the various thresholding values, the outlining and denoising presents clean data ready for analysis with object detection algorithms like FRCNN and YOLOv3.

KEYWORDS

Convolutional Matrix, Computer Vision, Machine Learning, Deep Learning, Automation Interface

1. INTRODUCTION

Much of the the hippocampal neuron body image data in the current scientific field contains unwanted data such as noise that is generated. In addition, neuron bodies that have neurons congregated together are unable to be segmented. Multiple neurons in close proximity are identified as a singular body. The current methodologies cause them to be grouped together. Smaller neurons with larger distances among themselves are easier to be segmented. In addition, some neuron data have different colors. Some images may contain a red-dominant color in the RGB spectrum, while others may contain a different dominance, and so forth.

Additionally, current filtration algorithms such as OpenCV's canny edge detection models only function on black and white images [1]. Other existing models such as the Sobel, Prewitt, and

Laplacian edge detection algorithms also leverage a gray-scale conversion while conducting edge detection processes [4, 5, 6]. The processes remove the color from the image, causing the image spectrum to be heavily manipulated. Our goal was to create an accurate model that can perform tasks despite the color gradients present within an image. Hence, our algorithms can process accurately on neuron data without changing the dynamics of the colors of the image.

As evident from current methods, disparities in hippocampal neuron data can cause future complexities for data analysis. When neuron data is close together, most algorithms fail to segment the neurons properly because of the proximities. Because of this, neurons are falsely grouped together with contours, leading to the failure for adaptive segmentation. In addition, unwanted data such as synapses are contoured. Without an effective denoising algorithm, such data will be processed in a further neural network. During the training of the neural network, the synapses will be used as a feature.

Although such issues relating to synapses may be avoidable in the status quo with cropping tools, the scientific field lacks a method that includes a cropping tool and data/image processing algorithms, all in one systematic interface.

Another dilemma in current image analysis methods include the singularity of data that it can process on. Images that contain a dominant color that is unsuitable for the algorithm is rendered useless for current algorithms. The RGB combination demonstrates a dominant color for neuron data. Some neuron stains contain red dominance (TRITC), while other stains such as the FITC stain contains a green dominance. With these disparities, current algorithms are not adaptive. To address this lack of modularity, we utilized user-inputted data that receives a threshold value that compares the image pixel intensities with. With this implementation, the difference in color dominance can be addressed.

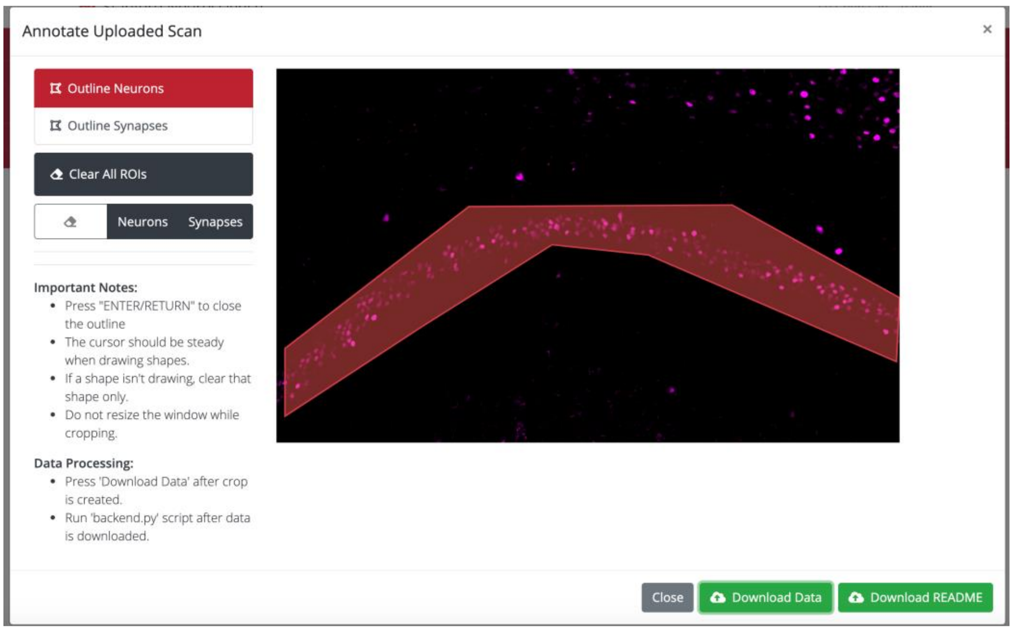
With potential problems in current image segmentation techniques, we propose a solution to address all scenarios. Our developed interface includes algorithms that can denoise unwanted data such as synapses, filter through images using convolution methods, and create contours for neuron data despite proximities with each other (disregarding groupings).

2. METHODS

The general methodology for image preprocessing that we employed starts through a web interface which allows for easy upload, labeling, masking, and cropping of ROI. After the images are cropped as targeted for their specific regions, a data.txt file is created containing the relevant information about image size and the specific points on the freehand cropped region. Due to resizing changes in the DOM as compared to original image files, the coordinates of the cropped points must be modulated in accordance to the new dimensions. Once a final correct masked region is created, it is ultimately ready to undergo subsequent denoising and edge detection to create general boundaries around each neuron.

2.1. GUI

Current tools for addressing image segmentation processes include cropping methods. Although images are able to cropped accurately, current available systematic methods lack the full-stack implementation to create crops, process the region of interest (ROI), and accurately contour the neurons in a single application. Using web development resources, we created a web app that includes a cropping tool keeping UX/UI in mind. With an input-image selector, one can input a given image for processing. After the respective crop is completed, one can download the neuron data to the local system. By executing a script, the neuron data (preferred threshold, dominant color, etc.) is taken into account, and the image is processed and contoured accurately in approximately 10 seconds on a machine with an Intel Core i5 processing unit. With an application that incorporates a variety of languages (web development and image processing), we developed an application that combines the resources for cropping and filtering/segmenting images in a customizable manner. Fig. 1 demonstrates an accurate representation of the web interface that is employed for the full-stack image segmentation tasks.



Important Notes:
- Press "ENTER/RETURN" to close the outline
- The cursor should be steady when drawing shapes.
- If a shape isn't drawing, clear that shape only.
- Do not resize the window while cropping.

Data Processing:
- Press 'Download Data' after crop is created.
- Run 'backend.py' script after data is downloaded.

Fig 1. A representation of the cropping tool with a generated crop around the region of interest (ROI). Erasures of different crops can be done via the different options presented.

2.2. Is on Edge

Traditional edge detection algorithms use implemented kernel methodologies to run a convolutional matrix throughout the image to create edges based on pixel intensity contrast. The algorithms present in this paper employ similar convolutional matrix methods however without the use of filtration and substituted with a novel checking system. The algorithm starts by analyzing a 3x3 pixel matrix of the image. The analysis is simply done on the central cell. It starts off by checking the presence of null information which occurs with low pixel intensities for each of the RGB values. For general purposes and through testing, we deemed an ideal value for such a threshold would be 50. Thus, depending on the stain the user is analyzing, the algorithm checks for null intensity on the appropriate RGB value (e.g. the TRITC channel which creates a red stain has a null intensity check for the R value of the pixel). If there exists a pixel in the 3 by 3 matrix other than the center pixel which has a pixel intensity for the respective channel that is less than 50, then it means the center pixel is in the presence of a cell with null intensity. Now, there is a second part for edge detection: it is to determine if the center pixel is in contact with a colored pixel (e.g for the TRITC channel, if the center pixel is in direct contact with a red pixel). This can be conclusively determined through thresholding. Once again, we deemed an ideal value for this would be 50 as well. Finally, if the center pixel is in contact with both a colored pixel and a pixel with null intensity, then it lies on the edge of a neuron. The general algorithm is presented below in pseudo code.

Line 1: denotes the pixel value intensity for the 3 by 3 cell matrix
```
pixels = [[i1, i2, i3],[i4, i5, i6],[i7, i8, i9]]
// i_n = the pixel color intensity for the appropriate light signal for
a specific pixel for i = 1,2,3...9.
```

Line 2, 3: defines threshold limits for null intensity and colored intensity
```
thresholdLower = 50 // Can be set by user thresholdUpper = 50 // Can be
set by user
```

Line 4: function definition for isOnEdge
```
function isOnEdge(pixels):
```

Line 5, 6: boolean variables for states of neighboring pixels
```
    doesNullIntensityExist = false doesColorIntensityExist = false
```

Line 7, 8: double for loop to parse through pixels array
```
    for row in pixels: for pixel in row:
```

Line 9, 10: Null intensity check
```
            if pixel < thresholdLower: doesNullIntesityExist = true
```

Line 11, 12: Color intensity check
```
            if pixel >
            thresholdUpper: doesColorIntesityExist = true
```

Line 13: returns the boolean function for both color and null pixel intensity existing.
```
    return doesNullIntensityExist and doesColorIntensityExist
```

2.3. Edge Detection

In the previous section, a general methodology for determining if a certain pixel resided on the edge of a neuron was outlined. In order to actually create a new image with the outlines of edges, this function must be run on each 3 by 3 pixel chunk of the image. If the function returns true, which indicates that the center pixel neighbors both a null intensity pixel and a colored pixel, the center pixel's color is changed to white to indicate that it is on the edge of a neuron. After running this function through each 3 by 3 section, a general edge outlines are created.

2.4. Denoising

After the general edges have been created, the data can be further augmented to relieve some of the additional noise. There are 3 specific ways of doing so: thinning out white borders, strengthening pixel intensities, and removing small regions.

2.4.1. Thinning Out White Borders

Once the edges have been created, there are regions where the border is multiple pixels wide. This starts to take away data from the original image. Thus, it is critical to thin out the white lines as much as possible; this is done in two steps. The first is to eliminate all the white pixels between the inner and outer border of the neuron edge. Essentially, an algorithm checks the 3 by 3 pixel cell matrix around a center cell and checks if all of them are the color of the edge (in our case white). If this is true, the cell color is returned to its original. After this process occurs each neuron is left with 2 borders: an inner one and an outer border. We then proceed to remove the inner border. This is done by simply checking for the presence of null intensity around the center pixel. Null intensity exists around the outer border but not in the inner one. This can eliminate the inner border. In addition, sometimes there exist lone white cells which are remnants of the thinning out white borders methods. This can also be removed by checking for the existence of null intensity around its neighboring pixels.

2.4.2. Strengthening Pixel Intensities

Once the borders have been thinned out, the image is ready to be further processed. In a lot of the hippocampal data that our lab worked with, there was additional noise present from faint colored synapses. To remove the synapses, a threshold method was used once again. To determine the threshold, we created a bucketing algorithm. The bucketing algorithm grouped all pixels based on their RGB intensity values. It created 10 buckets where each bucket represented intervals of length 25.6. These intervals contained the coordinates of cells with the specific light intensity channel within the end points of the appropriate interval. For example, if one was analyzing the red channel and the specific pixel had an intensity of 14 for its R value, that pixel was bucketed into the interval [0, 25.6]. Since the neuron’s and the synapse’s intensities differed by at least 50 (this was manually checked), the algorithm looked to find intervals where the quantity of bucketed pixels was smaller than the quantity of bucket pixels than its neighbors. This created an automatic way of determining the threshold pixel intensity value. Then, the image was parsed once more pixel by pixel. If the specific pixel has a lower value than the threshold for the appropriate channel, it was turned to black like the background; this removed a lot of noise. It essentially created a completely black background as opposed to a black background with subtle hints of red.

2.4.3. Removing Small Points

This algorithm is relatively straightforward. It takes 9x9 pixel matrices. The algorithm then proceeds to check if the boundary of these pixel matrices is entirely black. If so, this means that either the entire 9x9 section is black or there were small points that were entirely contained in this 9x9 interval, which need to be removed. If the edge is entirely black, then it blacks out every pixel in this 9x9 pixel matrix since it's either null information or small points. This works because all of the cells in our imaging data were much larger than a 9x9 pixel square matrix.

2.5. Data Transfer

In order to adequately incorporate a method that utilizes a cropping tool with the image analysis algorithms in a systematic full-stack tool, we developed a data transfer method to allow the script to process the image with the crop data that was generated from the web app. After data is downloaded from the web tool, the script processes and parses the input data in a correct format. Then, using our image segmentation algorithms and feature explorations, the image is processed within the same run time. The analyzed image with contours is exported to the local filesystem.

2.6. Data Parameterization

In order to custom fit the algorithms to user specification, the Python terminal interface contains convenient methodology for the addition of the various parameters. The code takes in the input paths to each of the images, a general folder path and naming convention for the various images, and finally thresholding values for each of the various filters. With these thresholding values, the user can custom fit the data augmentation to their needs. The user can determine the ideal threshold value with 8 runs (because there are 256 values for pixel intensity) using a binary search approach.

3. Results

The following (Fig. 2) describes the application of the edge detection algorithm on the Cy5 channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

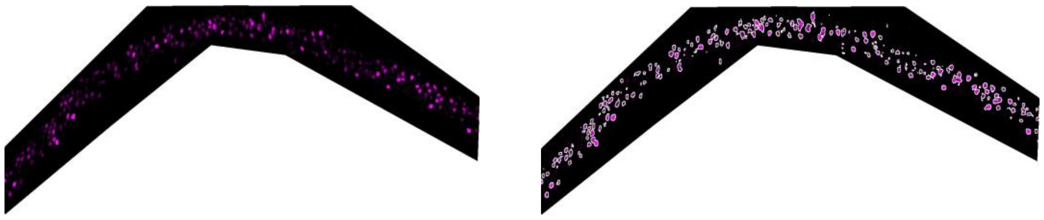


Fig 2. A side-by-side comparison of a before vs. after edge detection process on the Cy5 channel

The following (Fig. 3) describes the application of the edge detection algorithm on the DAPI channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

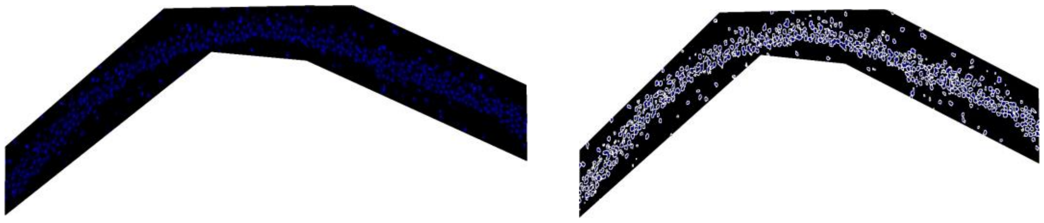


Fig 3. A side-by-side comparison of a before vs. after edge detection process on the DAPI channel

The following (Fig. 4) describes the application of the edge detection algorithm on the TRITC channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

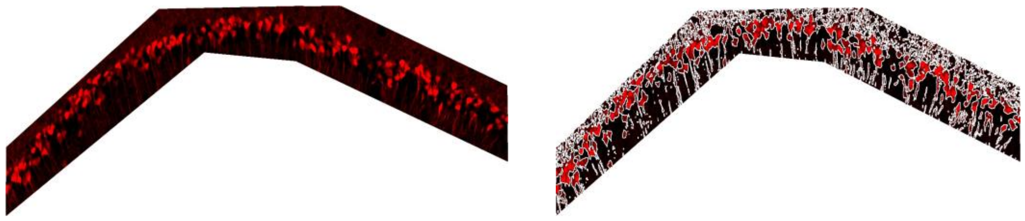


Fig 4. A side-by-side comparison of a before vs. after edge detection process on the TRITC channel

The following (Fig. 5) describes the application of the edge detection algorithm on the FITC channel stain. As evident, the white contours are able to distinguish between each neuron and the background.

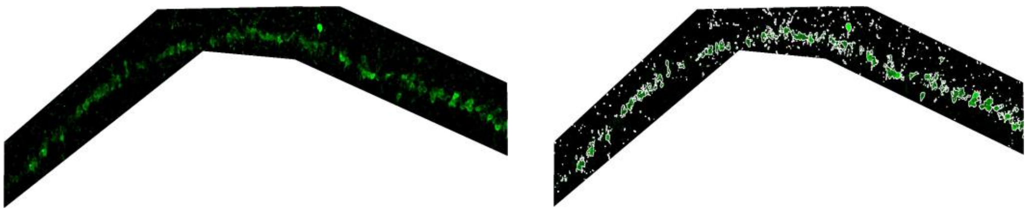


Fig 5. A side-by-side comparison of a before vs. after edge detection process on the FITC channel

The following (Fig. 6) describes the application of the white removal algorithm on the TRITC channel stain. As exhibited, the density of the white contours has been reduced.

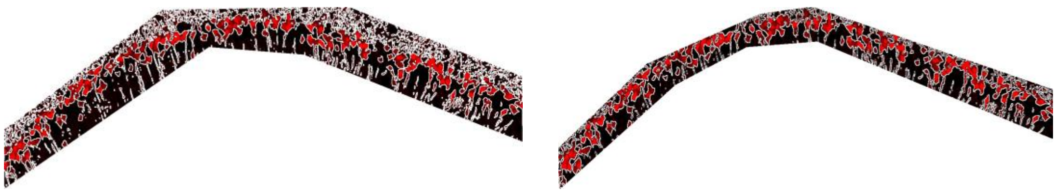


Fig 6. A side-by-side comparison of a before vs. after White Removal algorithm process on the TRITC channel

The following (Fig. 7) describes the application of the small point removal algorithm on the TRITC channel stain. As exhibited, the noise due to the synapses (small points) has been reduced. The final image at the bottom demonstrates the small point removal algorithm with the crop applied which has crisper quality as opposed to the original data.

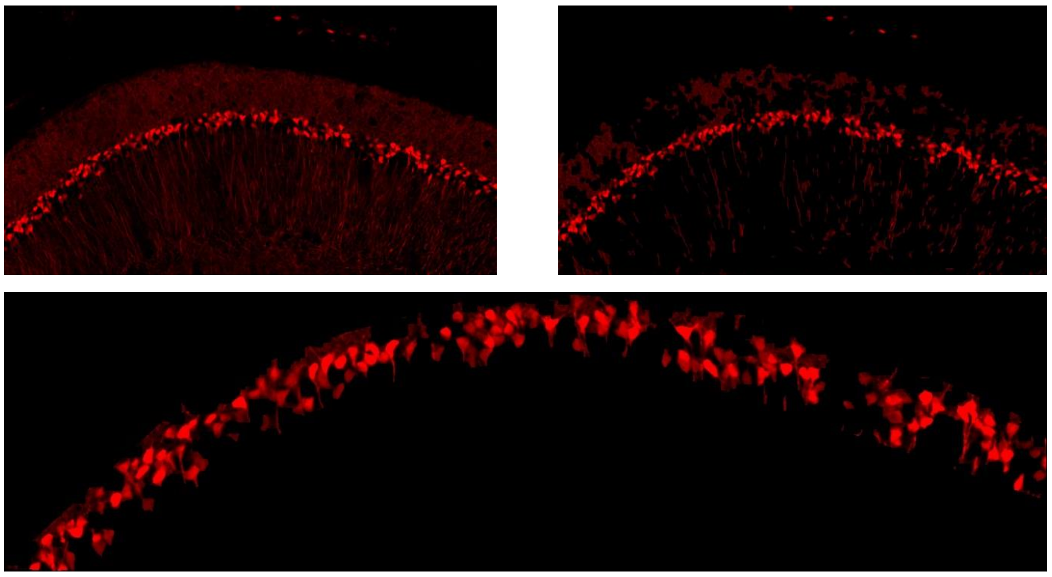


Fig 7. Top: A side-by-side comparison of a before vs. after Small Points Removal algorithm process on the TRITC channel

Fig 7. Bottom: Small Point Removal algorithm process on the TRITC channel With Crop

The following (Fig. 8) describes the application of the strengthen pixel intensity algorithm on the TRITC channel stain. Paired with the small point removal algorithm, this final data is the most clean as it contains thin borders/contours, an accurate crop, and little to none noise due to the synapses.

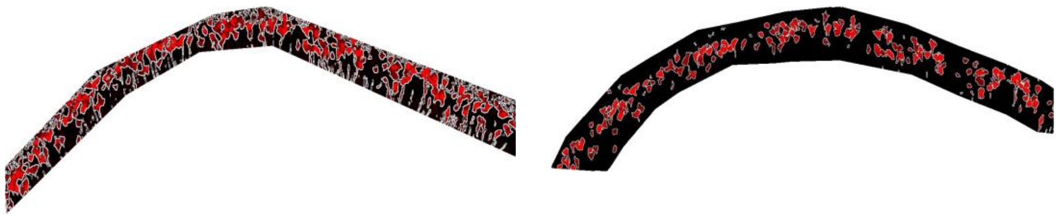


Fig 8. A Side-By-Side Comparison of A Before and After of the Strengthen Intensity Algorithm on TRITC Channel

4. CONCLUSIONS

In summary, this study demonstrates an algorithmic architecture that is capable of automating the segmentation of neuron data on various stains experimented on different hippocampus slices. In addition, the use of an interface to allow researchers and scientists to input image data in a robust method exhibited the increase in image segmentation techniques in the field of machine learning and automation. This approach provides sufficient resources for further analysis on images, for the contoured neurons can act as training data for future training or analysis.

ACKNOWLEDGEMENTS

We would like to thank our mentors, Professor Lu Chen and Dr. Bing Cao for guiding us through our project and helping us with the sufficient resources. In addition, we would like to thank Dr. Jie Li for providing us with our training and input data. We thank Dr. Amy Shon for providing us with access to GPU and computing resources such as the Sherlock cluster for us to train and run our models and matrices. Finally, we thank the Lu Chen lab from Stanford for providing us with the images and data that was adequate for conducting our research.

REFERENCES

[1] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/ TPAMI.1986.4767851.

[2] I I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," 2012 Proceedings of the 35th International Convention MIPRO, Opatija, 2012, pp. 1725-1730.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788

[4] Dey, D. and Polley, D., 2020. Edge Detection By Using Canny And Prewitt. [online] Ijser.org. Available at: <https://www.ijser.org/researchpaper/Edge-Detection-by-Using-Canny-and-Prewitt.pdf> [Accessed 25 June 2020].

[5] Owlnet.rice.edu. 2020. Laplacian Edge Detection. [online] Available at: <https:// www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html>

AUTHORS

Neeraj Rattehalli is a Bay Area high school student with a passion to automate the modern world. Growing up in a rapidly changing progressive society, Rattehalli has found machine learning and computational biology around every corner, and in order to stay steps ahead of the crowd, Rattehalli is conducting novel research at Stanford under the guidance of Lu Chen, Professor of Neuroscience. Rattehalli works with the intention and the zeal to solve today's most intricate challenges.

Ishan Jain is a Bay Area high school student with a passion in machine learning and computer vision. Ishan has worked on a variety of projects at Stanford Medicine, including analytical methods for assessing patients with peripheral artery disease (PAD) and developing mobile tools to create a remote surveillance application for postoperative surgery treatment using opioid medications. Furthermore, Ishan has worked with research professionals at the University of California, Santa Barbara, where he developed a meta-analysis tool to automate research manuscripts. Ishan is highly passionate about utilizing computer vision methods in application to the modern world.